1. Write a program to:

   - Read an int value from user input.
   - Assign it to a double (implicit widening) and print both.
   - Read a double, explicitly cast it to int, then to short, and print results—demonstrate truncation or overflow.

```
  package Assesement_day6;
public class implicit {

public static void main(String[] args) {
int intValue = 10;

double doubleValue = intValue;
System.out.println("Int value: " + intValue);
System.out.println("Double value: " + doubleValue);

double doubleInput = 1234567890.123;

int intCast = (int) doubleInput;
System.out.println("Double value: " + doubleInput);
System.out.println("Cast to int: " + intCast);

short shortCast = (short) doubleInput;
System.out.println("Cast to short: " + shortCast);

}
}
```

Output:

```
Int value: 10
Double value: 10.0
Double value: 1.234567890123E9
Cast to int: 1234567890
Cast to short: 722
```

2. Convert an int to String using String.valueOf(...), then back with Integer.parseInt(...). Handle NumberFormatException.

Compound Assignment Behaviour

1. Initialize int x = 5;.

2. Write two operations:

x = x + 4.5;   // Does this compile? Why or why not?

x += 4.5;   // What happens here?

```
package Assesement_day6;
public class Convert_int_string {
public static void main(String[] args) {
int x = 5;
x += 4.5;
System.out.println("integer=" +x);
String str = String.valueOf(x);
System.out.println("String=" + str);
int parseint = Integer.parseInt(str);
System.out.println("Parseint=" + parseint);

}

}
```

Output:

```
integer=9
String=9
Parseint=9
```

3. Print results and explain behavior in comments (implicit narrowing, compile error vs. successful assignment).

Object Casting with Inheritance

1. Define an Animal class with a method makeSound().
2. Define subclass Dog:
   - Override makeSound() (e.g. "Woof!").
   - Add method fetch().
3. In main:

```
Dog d = new Dog();
Animal a = d;                    // upcasting
a.makeSound();
```

```
package Assesement_day6;
class Animal {
void makeSound() {
System.out.println("The animal makes a sound.");
}
}

class Dog extends Animal {
void makeSound() {
System.out.println("Woof!");
}
void fetch() {
System.out.println("The dog fetches the cat.");
}
}

public class Animals_vs {

public static void main(String[] args) {
Dog d = new Dog();

Animal a = d;
a.makeSound();
if (a instanceof Dog) {
Dog d2 = (Dog) a;
d2.fetch();
}
}
}
```

Output:

```
Woof!

The dog fetches the cat
```

Mini- Project – Temperature Converter

1. **Prompt user for a temperature in Celsius (double).**
2. **Convert it to Fahrenheit:**

```
double fahrenheit = celsius * 9/5 + 32;
```

3. **Then cast that fahrenheit to int for display.**

```
package Assesement_day6;
import java.util.Scanner;

public class Temperature_celsius {

public static void main(String[] args) {
Scanner Scanner=new Scanner(System.in);
System.out.println("Enter temperature in celsius");
double celsius=Scanner.nextDouble();
double fahrenheit=(celsius*9/5)+32;
System.out.println(celsius + "°C is equal to " + fahrenheit +
"°F");
}

}
```

Output:

```
Enter temperature in celsius
15
15.0°C is equal to 59.0°F
```

4. Print both the precise (double) and truncated (int) values, and comment on precision loss

Enum

1: Days of the Week

Define an enum DaysOfWeek with seven constants. Then in main(), prompt the user to input a day name and:

- Print its position via ordinal().

Confirm if it's a weekend day using a switch or if-statement.

```
package Assesement_day6;

public class Enum {
Enum Day{Sunday,monday,tuesday,wednesday,thursday,friday,
saturday}

public static void main(String[] args) {
Day today = Day.Sunday;
switch(today) {
case Sunday:System.out.println("Sunday");
break;
case monday:System.out.println("monday");
break;
case tuesday:System.out.println("Sunday");
break;
case wednesday:System.out.println("wednsday");
break;
case thursday:System.out.println("thursday");
break;
case friday:System.out.println("friday");
break;
case saturday:System.out.println("saturday");
break;
}
if(today==Day.saturday || today==Day.Sunday){
System.out.println("weakend");
}
else {
System.out.println("weekday");
}
}
}
```

Output:

```
    Sunday
  weakend
```

2: Compass Directions

Create an enum Direction with the values NORTH, SOUTH, EAST, WEST. Write code to:

- Read a Direction from a string using valueOf().

Use switch or if to print movement (e.g. “Move north”). Test invalid inputs with proper error handling.

```
package Assesement_day6;
enum Direction {
NORTH, SOUTH, EAST, WEST
}

public class Directions {

public static void main(String[] args) {
String directionStr = "NORTH";
try {
Direction direction =
Direction.valueOf(directionStr.toUpperCase());
switch (direction) {
case NORTH:
System.out.println("Move north");
break;
case SOUTH:
System.out.println("Move south");
break;
case EAST:
System.out.println("Move east");
break;
case WEST:
System.out.println("Move west");
break;
}
} catch (IllegalArgumentException e) {
System.out.println("Invalid direction: " + directionStr);
}
}
}
```

Output:

```
Move north
```

3. Priority Levels with Extra Data

Implement enum PriorityLevel with constants (LOW, MEDIUM, HIGH, CRITICAL), each having:

- A numeric severity code.
- A boolean isUrgent() if severity ≥ some threshold.
  Print descriptions and check urgency.

```
package Assesement_day6;

enum PriorityLevel {

    LOW(0),MEDIUM(3),HIGH(7),CRITICAL(8);
    private final int severityCode;
    private static final int URGENCY_THRESHOLD = 6;

    PriorityLevel(int severityCode) {
        this.severityCode = severityCode;
    }
    public boolean isUrgent() {
        return severityCode >= URGENCY_THRESHOLD;
    }

    public int getSeverityCode() {
        return severityCode;
    }

    public String getDescription() {
        return name() + " (Severity Code: " + severityCode +
")";
    }
}
public class Security {
    public static void main(String[] args) {
        for (PriorityLevel level : PriorityLevel.values()) {
```

```
                        System.out.println(level.getDescription() + ", Urgent: " + level.isUrgent());
                }
        }
}
```

Output:

```
LOW (Severity Code: 0), Urgent: false
MEDIUM (Severity Code: 3), Urgent: false
HIGH (Severity Code: 7), Urgent: true
CRITICAL (Severity Code: 9), Urgent: true
```

4. Calculator Operations Enum

Create enum Operation (PLUS, MINUS, TIMES, DIVIDE) with an eval(double a, double b) method.
Implement two versions:

- **One using a switch(this) inside eval.**

  Another using constant-specific method overrides for eval. Compare both designs.

```
package Assesement_day6;

enum Operation {
}

    PLUS, MINUS, TIMES, DIVIDE;

    public double eval(double a, double b) {

        switch (this) {

            case PLUS:

                return a + b;

            case MINUS:

                return a - b;

            case TIMES:

                return a * b;

            case DIVIDE:

                if (b == 0) {

                    throw new
ArithmeticException("Cannot divide by zero");

                }

                return a / b;

            default:

                throw new RuntimeException("Invalid
operation");

        }

    }

}
```

```
public class Calculator {
    public static void main(String[] args) {
        System.out.println("10 + 5 = " +
Operation.PLUS.eval(10, 5));
        System.out.println("10 - 5 = " +
Operation.MINUS.eval(10, 5));
        System.out.println("10 * 5 = " +
Operation.TIMES.eval(10, 5));
        System.out.println("10 / 2 = " +
Operation.DIVIDE.eval(10, 2));
    }
}
```

Output:

```
10 + 5 = 15.0
10 - 5 = 5.0
10 * 5 = 50.0
10 / 2 = 5.0
```

Exception handling

1: Division & Array Access

Write a Java class ExceptionDemo with a main method that:

1. Attempts to divide an integer by zero and access an array out of bounds.

2. Wrap each risky operation in its own try- catch:
    - Catch only the specific exception types: ArithmeticException and ArrayIndexOutOfBoundsException.
    - In each catch, print a user-friendly message.
3. Add a finally block after each try- catch that prints "Operation completed.".

Example structure:

```
try {
        // division or array access
} catch (ArithmeticException e) {
        System.out.println("Division by zero is not allowed!");
} finally {
        System.out.println("Operation completed.");
}
```

```
public class ExceptionDemo {
        public static void main(String[] args) {
                      try {
                        int result = 10 / 0;
                } catch (ArithmeticException e) {
                        System.out.println("Division by zero is not allowed");
                } finally {
```

```
            System.out.println("Division operation
completed");
        }
        int[] array = new int[5];
        try {
            int value = array[10];
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index is out of
bounds");
        } finally {
            System.out.println("Array access operation
completed");
        }
    }
}
```

Output:

```
Division by zero is not allowed

Division operation completed

Array index is out of bounds

Array access operation completed
```

2: Throw and Handle Custom Exception

Create a class OddChecker:

1. Implement a static method:

```
public static void checkOdd(int n) throws
OddNumberException { /* ... */ }
```

2. If n is odd, throw a custom checked exception OddNumberException with message "Odd number: " + n.
3. In main:
   - Call checkOdd with different values (including odd and even).
   - Handle exceptions with try- catch, printing e.getMessage() when caught.

Define the exception like:

```
public class OddNumberException extends Exception {
    public OddNumberException(String message)
{ super(message); }
}
```

```
package Assesement_day6;
class OddNumberException extends Exception {
public OddNumberException(String message) {
super(message);
}
}

public class OddChecker {
```

```
public static void checkOdd(int n) throws
OddNumberException {
if (n % 2 != 0) {
throw new OddNumberException("Odd number: " + n);
} else {
System.out.println(n + " is even");
}
}
public static void main(String[] args) {
int[] numbers = {10, 23, 44, 57, 92};

for (int number : numbers) {
try {
checkOdd(number);
} catch (OddNumberException e) {
System.out.println(e.getMessage());
}
}
}
}
```

Output:

```
10 is even
Odd number: 23
44 is even
Odd number: 57
92 is even
```

3.File Handling with Multiple Catches

Create a class FileReadDemo:

1. In main, call a method readFile(String filename) that declares throws FileNotFoundException, IOException.
2. In readFile, use FileReader (or BufferedReader) to open and read the first line of the file.
3. Handle exceptions in main using separate catch blocks:
   - catch (FileNotFoundException e) → print "File not found: " + filename
   - catch (IOException e) → print "Error reading file: " + e.getMessage()"
4. Include a finally block that prints "Cleanup done." regardless of outcome.

```
package Assesement_day6;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FileReadDemo {
public static void readFile(String filename) throws
FileNotFoundException, IOException {
try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
String line = reader.readLine();
System.out.println("First line of the file: " + line);
}
}

public static void main(String[] args) {
```

```
String filename = "data.txt";

try {
readFile(filename);
} catch (FileNotFoundException e) {
System.out.println("File not found: " + filename);
} catch (IOException e) {
System.out.println("Error reading file: " + e.getMessage());
} finally {
System.out.println("Cleanup done");
}
}
}
```

Output:

```
First line of the file: Welcome to Training!
Cleanup done
```

4: Multi- Exception in One Try Block

Write a class MultiExceptionDemo:

- In a single try block, perform:
  - Opening a file
  - Parsing its first line as integer
  - Dividing 100 by that integer
- Use multiple catch blocks in this order:

1. FileNotFoundException
2. IOException

3. NumberFormatException

4. ArithmeticException

```
package Assesement_day6;

public class Nullpoint_exception {
public static void main(String[] args) {
try {
String str = null;
System.out.println(str.length());
}
catch (NullPointerException e) {
System.out.println("NullPointerException");
}
try {
Object obj=5 ;
String s = (String) obj;
}
catch (ClassCastException e) {
System.out.println("ClassCastException");
}
try {
Class.forName("Nodatafound");
}
catch (ClassNotFoundException e) {
System.out.println("ClassNotFoundException");
}
try {
String s="123";
}
catch (IllegalArgumentException e ) {
```

```
System.out.println("IlligelArgumentException");
}
try {
String str = "abc";
int num = Integer.parseInt(str);
}
catch (NumberFormatException e) {
System.out.println("NumberFormatException");
}
}
}
```

Output:

```
NullPointerException
ClassCastException
ClassNotFoundException
NumberFormatException
```